Imperial College London

Imperial College London

Department of Electrical and Electronic Engineering

Real-time Obstacle Detection for High-speed Railways Using Deep Neural Networks Based on FPGAs

Author:
Tsz-hang Wong
Haoran Wu

Supervisor:
Dr James J. Davis

UROP Report

September 24, 2022

Abstract

With the increasing speed of railways, an obstacle detection method with extremely low latency is required to ensure the train has enough braking distance to stop. The application of convolutional neural networks (CNNs) is proposed as one of the potential and viable methods. Although there are various CNN-based algorithms for real-time obstacle detection, they often result in large models or high computational complexity which are not regarded as efficient implementations. Therefore, we focusses on the construction of neural network hardware accelerators on FPGAs, exploiting various optimization techniques and strategies.

Introduction & Preliminary

You only look once(Yolo), a real-time object detection system, was elected as the baseline system for our application. It surpasses other systems like R-CNN in terms of computational complexity since each prediction for a single image is only required to go through a single network evaluation once, unlike other systems which require thousands of recursions. This makes it 1000× faster than R-CNN and 100× faster than Fast R-CNN, reaching the state-of-art performance for real-time object detection [1]. In our framework, we started off by using Tiny-Yolov3, which is the simplified version of Yolov3 with fewer convolutional layers in its backbone classifier compared with Yolov3, aiming to reduce the computational complexity.

To make the system process image even faster without exerting a negative impact on its accuracy and accelerator scalability, we adopted the backbone classifier with the Residual Binary Network (ReBNet) architecture, a binary neural network with multiple levels of residual binarization [2].

In comparison with the general binarized models provided for image classification, the yolov3 used multi-scale prediction classifiers, involving new operations like skip connections, concatenations, and resizing that were previously not supported by the RebNet model in terms of either hardware or software [1]. In more detail, after the darknet [1], the output tensor was cloned into two tensors with one continuing to go deeper into the network while the other one was preserved for later use. These two tensors were then concatenated together at the nearest end of the network.

As area efficiency also comes into concern, certain layers were then changed to the Lookup Table Network (LUTNet) architecture, an FPGA-based neural network accelerator using native LUTs as inference operators. The LUTNet model introduces nonlinearity to the network by replacing the main function in CNN with function $\mathrm{g}\left(\tilde{\mathrm{x}}^{(\mathrm{n})}\right)$ where $\tilde{x}$ are any K components of the original input vector x as follows:

$$y = f\left(\sum_{n=1}^{N} \omega_n x_n\right) \qquad (1) \qquad\qquad y = f\left(\sum_{n=1}^{\tilde{N}} g_n\left(\tilde{x}^{(n)}\right)\right) \qquad (2)$$

with each $\mathrm{g}\left(\tilde{\mathrm{x}}^{(\mathrm{n})}\right)$ implemented in a LUT [3]. Therefore, far heavier pruning, the removal of low weighted network connections, can be done by taking benefit from the nonlinear Boolean functions and flexibility of the LUTs, resulting the size of the adder tree ($\tilde{\mathrm{N}} \ll \mathrm{N}$) for summation and achieving significant area saving [3]. For demonstration purposes, we have chosen to exploit the K=4 case, where K represents the number of inputs for each LUT. To enable fast convergence and avoid overfitting, the weights were initialized in the way proposed by [3], setting the start point of the weight to the pre-trained model by equating the non-pruned with the interpolating function of $\mathrm{g}\left(\tilde{\mathrm{x}}^{(\mathrm{n})}\right)$. The model means average precision (mAP), which is a parameter representing the quality of detection [1], with the combined 4-LUTs models was shown to be approximately the same.

By combining and utilizing all these frameworks provided, we constructed a 4-LUT-based Yolov3 model and its corresponding hardware FPGA-based neural network accelerators.

Related Work

Our working progress is demonstrated in the following flow chart. Due to time limitations, we did not complete all the tasks we planned. We focused on training our RebNet Model and implementing it onto the FPGA board. The uncompleted tasks are linked by a dashed arrow in the chart. Our detailed work is discussed below.

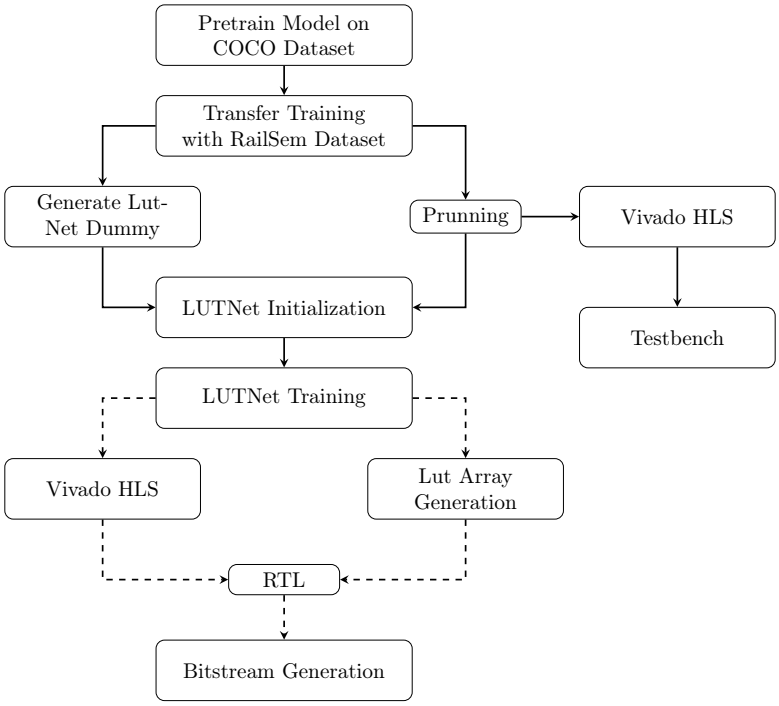

Figure 1: Task Flow Chart

• Training Process

With regard to the training dataset, Railsem Dataset is selected as the main resource for neural network training. However, due to the lack of the dataset for specific classes within this database, the accuracy of the detection does not meet our requirements. We improved this by conducting transfer learning. The weight was initially trained with the COCO dataset and then transferred to the training with the RAILSEM dataset. During the later training, low level layers (layer index: 0 to I) were fixed to preserve low-level features extracted from the COCO for the first few epochs. The fixed layers were then gradually released to be trainable along with the training process. After experimenting with different index I and how layers are being released, one satisfying result was obtained by setting I=5 and with releasing order in Appendix(A). Apart from that, some trials in modifying the model's hyperparameter involving doubling channels, adding bias for certain layers, and adding an l2 regularizer [3], were carried out but had little effect on improving the accuracy.

The initial training framework we used for the binaried network can be divided into two steps. Firstly, the model was trained with real number weights clipped to [-1,1]. This forces the weight to scatter around 1 and -1. Furthermore, as long as the loss was minimized, the weights could then be binarized to -1, 1 with a straight-through estimator added for further training. However, with quantized weights, the model can only output quantized values, causing 0 widths or 0 height in inferred bounding boxes. This leads to undefined behaviour in calculating the intercept and union (IOU) loss, outputting NaN. To address this problem, we simply substituted the inferred height and width with the minimum value observed from the real number model.

This method is viable based on the assumption that the model is already fitting the data quite well, otherwise, this substituting operation might cause the gradient descent to the wrong direction or even worse, lead to gradient explosion. Hence, to preserve the model's accuracy as much as possible before and after binarization, two quantization values β, α instead of -1,1 could be found and used,

such that the total standard variance of difference in weights before and after quantization reaches a minimum. After the model converged, the β and α could always be converted back to -1 or 1 by scaling up by a factor S or adding an offset by O along with the residual level-wise scaling factors γ in ReBNet changed to $\gamma/S - \gamma * O/S$ due to linearity. This also leads to faster convergence, since the starting point is closer to the global minima. In our case, we observed the weights are scattered more around 0 hence we tried out setting S to be 0.5 and O to be 0 and that was enough for the model to converge.

• Configuration Modification

The related current research mainly focussed on the realization of image classification rather than object detection on an FPGA board [3]. To adapt our chosen TinyYolov3 model onto the FPGA, we did the following modifications based on the provided RebNet code.

To begin with, we added several padding options for each layer into the hardware library to meet the software model description. Since data transfers between layers can only be a combination of residual levels' value, pixels with 0 value can not be parsed. We added a signal to record the current output pixel number. The sliding window inputs of any output pixels located at the image edges would enter a padding check before performing the XNOR operation. For example, one pixel with all its channels having values 0 are determined as a padding pixel and excluded from the PopCount operation (adding half of the layer's fanins to the Popcount unit).

New layers were designed to cope with concatenation followed by a convolution operation. Since the inputs needed to be concatenated together come from two different layers, they share different parameters like fanins, residual weights, SIMD, and PE leading to different coefficients used in MVTU. Hence, the convolution was performed separately with the input first and the two accumulated results were then simply summed up and passed through a single threshold (sum of the two thresholds) to achieve the same concatenation effect.

Apart from that, we also implemented a max pooling layer supporting stride one. The input matrix was first half padded with the padding option then values in any 2 x 2 windows of the feature maps are obtained by going through SWU with stride 1. After selecting the max value within the results, we write it to the output. A 2 $\times$ resizing unit is also implemented to meet the requirement of the model.

In terms of the output of the network, we modified the binarized convolution layer by removing its activation, to provide the non-binarized result. It enables us to decode the output of a neural network.

• Testbench

To debug our model on the hardware side, following the procedure suggested by an MSC student, we first extracted the feature map tensor before and after the specific layer, we want to test on the software side and save them into a set of txt files. Then we let Csim read those files and transfer them into streaming input of the specific layers in HLS hardware. The results obtained on both sides are then compared to verify the functionality of the layer.

Experiment Result

Due to the fact that training such a large model is time-consuming, we capped the training epochs to 35. Epoch represents the number of iterations when an entire dataset is passed forward and backward through the neural network. Also, we noticed that since the model was quantized, it performs badly in detecting small objects. Therefore, classes of small objects such as poles or classes with limited training data were excluded from the RAILSEM dataset before training for better accuracy.

According to the experiment, models with double channels generally exhibit much higher accuracy than a single one. However, since the doubling channel would cover twice memory, due to the limitation of GPU memory, training doubled channel would lead to GPU resource exhaustion, despite the fact that we have decreased batch size to one and use multiple GPU to train the model. Apart from that, we found that using bias in the output convolutional layer has little effect on the

training result. Therefore, considering the complexity and scalability of our model, we finally take a single channel without adding bias to proceed with our implementation on the hardware side.

In terms of binarizing our models, according to the result, the accuracy of the model dropped by 50%, which is still within our expectations. That is because binarization would largely affect the accuracy of the model.

Table 1: Training Result

Model Architecture	Epoch 25 mAP	Epoch 35 mAP
Full Precision Single Chan	7.00% (14 classes)	10.00% (14 classes)
Full Precision Single Chan With Bias	10.00% (14 classes)	18.00% (8 classes)
Full Precision Double Chan	16.40% (8 classes)	17.70% (8 classes)
Full Precision Double Chan With Bias	10.00% (8 classes)	16.50% (8 classes)
Binarized Single Chan	N/A	5.34% (7 classes)
Binarized Double Chan	N/A	11.00% (7 classes)

Reflective Component

Through the nine weeks of research in the field of BNN, we are able to have a brief understanding of its principles. We also learned how to train and verify our models. We had the opportunity to investigate and tried out various methods to optimize the training result. In the training process, we managed to tackle many challenging problems like gradient explosion.

Apart from that, in terms of the hardware side, we spent a long time modifying and debugging the provided BNN repositories to fit our model, which enables us to think in the hardware perspective and also design hardware with Vivado HLS. Also, to perform testing on our model, we investigated an MSC student's testbench and also adopted it to our model. We learned how to write a testbench for hardware with Csim.

In the future, we are considering implementing LUTNet on the hardware side. Apart from that, we want to try a better neural network model with higher accuracy in image detection.

In conclusion, we believe that what we learned through this UROP built a concrete foundation in the field of BNN and FPGA for us, which would be really helpful in our future work. We both developed great interest in this field and are willing to carry on our study in this field in future work.

Reference

[1] W. Gai, Y. Liu, J. Zhang, and G. Jing, "An improved tiny yolov3 for real-time object detection," *Systems Science & Control Engineering*, vol. 9, no. 1, pp. 314–321, 2021. [Online]. Available: https://doi.org/10.1080/21642583.2021.1901156

[2] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "Rebnet: Residual binarized neural network," 2017. [Online]. Available: https://arxiv.org/abs/1711.01243

[3] E. Wang, J. J. Davis, P. Y. K. Cheung, and G. A. Constantinides, "LUTNet: Learning FPGA configurations for highly efficient neural network inference," *IEEE Transactions on Computers*, 2020, to appear.